

Hugues Bersini

6^e édition

La programmation orientée objet

Cours et exercices en UML 2
avec Java, C#, C++, Python, PHP et LINQ

EYROLLES

Avant-propos

Aux tout débuts de l'informatique, le fonctionnement interne des processeurs décidait de la seule manière efficace de programmer un ordinateur. Alors que l'on acceptait tout programme comme une suite logique d'instructions, il était admis que l'organisation du programme et la nature même de ces instructions ne pouvaient s'éloigner de la façon dont le processeur les exécutait : pour l'essentiel, des modifications de données mémorisées, des déplacements de ces données d'un emplacement mémoire à un autre, des opérations d'arithmétique et de logique élémentaire, et de possibles ruptures de séquence ou branchements.

La mise au point d'algorithmes complexes, dépassant les simples opérations mathématiques et les simples opérations de stockage et de récupérations de données, obligea les informaticiens à effectuer un premier saut dans l'abstrait, en inventant un style de langage dit procédural, auquel appartiennent les langages Fortran, Cobol, Basic, Pascal, C... Les codes écrits dans ces langages sont devenus indépendants des instructions élémentaires propres à chaque type de processeur. Grâce à eux, les informaticiens ont pris quelques distances par rapport aux processeurs (en ne travaillant plus directement à partir des adresses mémoire et en évitant la manipulation directe des instructions élémentaires) et ont élaboré une écriture de programmes plus proche de la manière naturelle de poser et de résoudre les problèmes. Il est incontestablement plus simple d'écrire : $c = a + b$ qu'une suite d'instructions telles que "load a, reg1", "load b, reg2", "add reg3, reg1, reg2", "move c, reg3", ayant pourtant la même finalité. Une opération de traduction automatique, dite de compilation, se charge de traduire le programme, écrit au départ avec ces nouveaux langages, dans les instructions élémentaires seules comprises par le processeur.

La montée en abstraction permise par ces langages de programmation présente un double avantage : une facilité d'écriture et de résolution algorithmique, ainsi qu'une indépendance accrue par rapport aux différents types de processeur existant aujourd'hui sur le marché. Le programmeur se trouve libéré des détails d'implémentation machine et peut se concentrer sur la logique du problème et ses voies de résolution.

Plus les problèmes à affronter gagnaient en complexité – comptabilité, jeux automatiques, compréhension et traduction des langues naturelles, aide à la décision, bureautique, conception et enseignement assistés, programmes graphiques, etc. –, plus l'architecture et le fonctionnement des processeurs semblaient contraignants. Il devenait vital d'inventer des mécanismes informatiques simples à mettre en œuvre pour réduire cette complexité et rapprocher encore plus l'écriture de programmes des manières humaines de poser et résoudre les problèmes.

Avec l'intelligence artificielle, l'informatique s'inspira de notre mode cognitif d'organisation des connaissances, comme un ensemble d'objets conceptuels entrant dans un réseau de dépendance et se structurant de manière taxonomique. Avec la systémique ou la bioinformatique, l'informatique nous révéla qu'un ensemble d'agents au fonctionnement élémentaire, mais s'influençant mutuellement, peut produire un comportement émergent d'une surprenante complexité lorsqu'on observe le système dans sa globalité. Dès lors, pour comprendre jusqu'à reproduire ce comportement par le biais informatique, la meilleure approche consiste en une découpe adéquate du système en ses parties et une attention limitée au fonctionnement de chacune d'entre elles.

Tout cela mis ensemble (la nécessaire distanciation par rapport au fonctionnement du processeur, la volonté de rapprocher la programmation du mode cognitif de résolution de problème, les percées de l'intelligence artificielle et de la bio-informatique, le découpage comme voie de simplification des systèmes apparemment complexes) conduisit graduellement à un deuxième type de langages de programmation, fêtant ses 45 ans d'existence (l'antiquité à l'échelle informatique) : les langages orientés objet, tels Simula, Smalltalk, C++, Eiffel, Java, C#, Delphi, Power Builder, Python et bien d'autres...

L'orientation objet (OO) en quelques mots

À la différence de la programmation procédurale, un programme écrit dans un langage objet répartit l'effort de résolution de problèmes sur un ensemble d'objets collaborant par envoi de messages. Chaque objet se décrit par un ensemble d'attributs (partie statique) et de méthodes portant sur ces attributs (partie dynamique). Certains de ces attributs étant l'adresse des objets avec lesquels les premiers coopèrent, il leur est possible de déléguer certaines des tâches à leurs collaborateurs. Le tout s'opère en respectant un principe de distribution des responsabilités on ne peut plus simple, chaque objet s'occupant de ses propres attributs. Lorsqu'un objet exige de s'informer sur les attributs d'un autre ou de les modifier, il charge cet autre de s'acquitter de cette tâche. En effet, chaque objet expose à ses interlocuteurs un mode d'emploi restreint, une carte de visite limitée aux seuls services qu'il est apte à assurer et continuera à rendre dans le temps, malgré de possibles modifications dans la réalisation concrète de ces services.

Cette programmation est fondamentalement distribuée, modularisée et décentralisée. Pour autant qu'elle respecte également des principes de confinement et d'accès limité (dits d'encapsulation, l'objet n'expose qu'une partie restreinte de ses services), cette répartition modulaire a également l'insigne avantage de favoriser la stabilité des développements. En effet, elle restreint au maximum les conséquences de modifications apportées au code au cours du temps : seuls les objets concernés sont modifiés, pas leurs interlocuteurs, même si le comportement de ces derniers dépend indirectement des fonctionnalités affectées.

Ces améliorations, résultant de la prise de conscience des problèmes posés par l'industrie du logiciel (complexité accrue et stabilité dégradée), ont enrichi la syntaxe des langages objet. Un autre mécanisme de modularisation inhérent à l'orienté objet est l'héritage, qui permet à la programmation de refléter l'organisation taxonomique de notre connaissance en une hiérarchie de concepts du plus au moins général. À nouveau, cette organisation modulaire en objets génériques et plus spécialistes est à

l'origine d'une simplification de la programmation, d'une économie d'écriture et de la création de zones de code aux modifications confinées. Tant cet héritage que la répartition des tâches entre les objets autorisent une décomposition plus naturelle des problèmes, une réutilisation facilitée des codes déjà existants (tout module peut se prêter à plusieurs assemblages) et une maintenance facilitée et allégée de ces derniers. L'orientation objet s'impose, non pas comme une panacée universelle, mais comme une évolution naturelle de la programmation procédurale qui facilite l'écriture de programmes, les rendant plus gérables, plus compréhensibles, plus stables et mieux réexploitables.

L'orienté objet inscrit la programmation dans une démarche somme toute très classique pour affronter la complexité de quelque problème qui soit : une découpe naturelle et intuitive en des parties plus simples. A fortiori, cette découpe sera d'autant plus intuitive qu'elle s'inspire de notre manière « cognitive » de découper la réalité qui nous entoure. L'héritage, reflet fidèle de notre organisation cognitive, en est le témoignage le plus éclatant. L'approche procédurale rendait cette découpe moins naturelle, plus « forcée ». Si de nombreux adeptes de la programmation procédurale sont en effet conscients qu'une manière incontournable de simplifier le développement d'un programme complexe est de le découper physiquement, ils souffrent de l'absence d'une prise en compte naturelle et syntaxique de cette découpe dans les langages de programmation utilisés. Dans un programme imposant, l'OO aide à tracer les pointillés que les ciseaux doivent suivre là où il semble le plus naturel de les tracer : au niveau du cou, des épaules ou de la ceinture, et non pas au niveau des sourcils, des biceps ou des mollets. De surcroît, cette pratique de la programmation incite à cette découpe suivant deux dimensions orthogonales : horizontalement, les classes se déléguant mutuellement un ensemble de services, verticalement, les classes héritant entre elles d'attributs et de méthodes installés à différents niveaux d'une hiérarchie taxonomique. Pour chacune de ces dimensions, reproduisant fidèlement nos mécanismes cognitifs de conceptualisation, en plus de simplifier l'écriture des codes, il est important de faciliter la récupération de ces parties dans de nouveaux contextes et d'assurer la robustesse de ces parties aux changements survenus dans d'autres. Un code OO, idéalement, sera aussi simple à créer qu'à maintenir, récupérer et faire évoluer.

Il n'est pas pertinent d'opposer le procédural à l'OO car, in fine, toute programmation des méthodes (c'est-à-dire la partie active des classes et des objets) reste totalement tributaire des mécanismes procéduraux. On y rencontre des variables, des arguments, des boucles, des fonctions et leurs paramètres, des instructions conditionnelles, tout ce que l'on trouve classiquement dans les boîtes à outils procédurales. L'OO vient plutôt compléter le procédural, en lui superposant un système de découpe plus naturel et facile à mettre en œuvre. Pour preuve, les langages procéduraux comme le C, Cobol ou, plus récemment, PHP, se sont relativement aisément enrichis d'une couche dite OO sans que cette addition ne remette sérieusement en question l'existant.

Cependant, l'effet de cette couche additionnelle ne se limite pas à quelques structures de données supplémentaires afin de mieux organiser les informations manipulées par le programme. Il va bien au-delà. C'est toute une manière de concevoir un programme et la répartition de ses parties fonctionnelles qui est en jeu. Les fonctions et les données ne sont plus d'un seul tenant mais éclatées en un ensemble de modules reprenant chacun à son compte une sous-partie de ces données et les seules fonctions qui les manipulent. Il faut réapprendre à programmer en s'essayant au développement d'une succession de micro-programmes et au couplage soigné et réduit au minimum de ces micro-programmes.

En découpant 1 000 lignes de code en 10 modules de 100 lignes, le gain est bien plus que linéaire, car il est extraordinairement plus simple de programmer 100 lignes plutôt que 1 000. En subs-

tance, la programmation OO pourrait reprendre à son compte ce slogan altermondialiste : « agir localement, penser globalement ».

Se pose alors la question de tactique didactique, très controversée dans l'enseignement de l'informatique aujourd'hui, sur l'ordre dans lequel enseigner procédural et OO. De nombreux enseignants, soutenus en cela par de très nombreux manuels, considèrent qu'il faut d'abord passer par un enseignement intensif et une maîtrise parfaite du procédural, avant de faire le grand saut vers l'OO. Mais vingt années d'enseignement de la programmation à des étudiants de tous âges et de toutes conditions (issus des sciences humaines ou exactes) nous ont convaincus qu'il n'y a aucun ordre à donner. De même qu'historiquement, l'OO est né quasiment en même temps que le procédural et en complément de celui-ci, l'OO doit s'enseigner conjointement et en complément du procédural. Il faut enseigner les instructions de contrôle en même temps que la découpe en classes. L'enseignement de la programmation doit mélanger à loisir la perception « micro » des mécanismes procéduraux à la vision « macro » de la découpe en objets. Aujourd'hui, tout projet informatique de dimension conséquente débute par une analyse des différentes classes qui le constituent. Il faut aborder l'enseignement de la programmation tout comme débute la prise en charge de ce type de projet, en enseignant au plus vite la manière dont ces classes et les objets qui en résultent opèrent à l'intérieur d'un programme.

Ces dernières années, compétition oblige, l'orienté objet s'est trouvé à l'origine d'une explosion de technologies différentes, mais toutes intégrant à leur manière ses mécanismes de base : classes, objets, envois de messages, héritage, encapsulation, polymorphisme... Ainsi sont apparus de nombreux langages de programmation proposant des syntaxes dont les différences sont soit purement cosmétiques, soit plus subtiles. Ils sont autant de variations sur les thèmes créés par leurs trois principaux précurseurs : Simula, Smalltalk et C++.

L'OO a également conduit à repenser trois des chapitres les plus importants de l'informatique de ces deux dernières décennies :

- tout d'abord, le besoin de développer une méthode de modélisation graphique standardisée débouchant sur un niveau d'abstraction encore supplémentaire (on ne programme plus en écrivant du code mais en dessinant un ensemble de diagrammes, le code étant créé automatiquement à partir de ceux-ci ; c'est le rôle joué par UML 2) ;
- ensuite, les applications informatiques distribuées (on ne parlera plus d'applications distribuées mais d'objets distribués, et non plus d'appels distants de procédures mais d'envois de messages à travers le réseau) ;
- enfin, le stockage des données, qui doit maintenant compter avec les objets.

Chaque fois, plus qu'un changement de vocabulaire, un changement de mentalité sinon de culture s'impose.

Les grands acteurs de l'orienté objet

Aujourd'hui, l'OO est omniprésent. Microsoft par exemple, a développé un nouveau langage informatique purement objet (C#), a très intensément contribué au développement d'un système d'informatique distribuée, basé sur des envois de messages d'ordinateur à ordinateur (les services web) et a plus

récemment proposé un nouveau langage d'interrogation des objets (LINQ), qui s'interface naturellement avec le monde relationnel et le monde XML. Tous les langages informatiques intégrés dans sa nouvelle plate-forme de développement, .Net (aux dernières nouvelles, ils seraient 22), visent à une uniformisation (y compris les nouvelles versions de Visual Basic et Visual C++) en intégrant les mêmes briques de base de l'OO. Aboutissement considérable s'il en est, il devient très simple de faire communiquer ou hériter entre elles des classes écrites dans des langages différents.

Plusieurs années auparavant, Sun avait conçu Java, une création déterminante car elle fut à l'origine de ce nouvel engouement pour une manière de programmer qui pourtant existait depuis toujours sans que les informaticiens dans leur ensemble en reconnaissent l'utilité ni la pertinence. Depuis, Sun a créé RMI, Jini et sa propre version des services web, tous basés sur les technologies OO. Ces mêmes services web font l'objet de développements tout autant aboutis chez HP ou IBM. À la croisée de Java et du Web (originellement la raison, sinon du développement de Java, du moins de son succès), on découvre une importante panoplie d'outils de développement et de conception de sites web dynamiques. Depuis, Java est devenu le langage de prédilection pour de nombreuses applications d'entreprise et plus récemment pour le développement d'applications tournant sur les smartphones et tablettes dotés du système Android, maintenu par Google.

IBM et Borland, avec Rational et Together, menaient la danse en matière d'outils d'analyse du développement logiciel, avec la mise au point de puissants environnements UML. Chez IBM, la plate-forme logicielle Eclipse est sans doute, à ce jour, une des aventures Open Source les plus abouties en matière d'OO. Comme environnement de développement Java, Eclipse est aujourd'hui le plus prisé et le plus usité et cherche à gagner son pari « d'éclipser » tous les autres. Borland a rendu Together intégrable tant dans Visual Studio.Net que dans Eclipse, comme outil synchronisant au mieux et au plus la programmation et la réalisation des diagrammes UML.

Enfin, l'OMG, organisme de standardisation du monde logiciel, n'a pas pour rien la lettre O comme initiale. UML et Corba sont ses premières productions : la version OO de l'analyse logicielle et la version OO de l'informatique distribuée. Cet organisme plaide de plus en plus pour un développement informatique détaché des langages de programmation ainsi que des plates-formes matérielles, par l'utilisation intensive des diagrammes UML. Partant de ces mêmes diagrammes, les codes seraient créés automatiquement dans un langage choisi et en adéquation avec la technologie voulue.

Le pari d'UML est osé et encore très largement controversé, mais l'évolution de l'informatique au cours des ans a toujours confié à des mécanismes automatisés le soin de prendre en charge des détails qui éloignaient le programmeur de sa mission première : penser et résoudre son problème.

Objectifs de l'ouvrage

Toute pratique économe, fiable et élégante de Java, C++, C#, Python, PHP ou UML requiert, pour débiter, une bonne maîtrise des mécanismes de base de l'OO. Et, pour y parvenir, rien n'est mieux que d'expérimenter les technologies OO dans ces différentes versions, comme un bon conducteur qui se sera frotté à plusieurs types de véhicules, un bon skieur à plusieurs styles de skis et un guitariste à plusieurs modèles de guitares.

Plutôt qu'un voyage en profondeur dans l'un ou l'autre de ces multiples territoires, ce livre vous propose d'explorer plusieurs d'entre eux, mais en tentant à chaque fois de dévoiler ce qu'ils recèlent de commun. Car ce sont ces ressemblances qui constituent les briques fondamentales de l'OO. Nous pensons que la mise en parallèle de C++, Java, C#, Python, PHP 5 et UML est une voie privilégiée pour l'extraction de ces mécanismes de base.

Il nous a paru pour cette raison indispensable de discuter et comparer la façon dont ces cinq langages de programmation gèrent, par exemple, l'occupation mémoire par les objets, leur manière d'implémenter le polymorphisme ou la programmation dite « générique », pour en comprendre in fine toute la problématique et les subtilités indépendamment de l'une ou l'autre implémentation. Ajoutez une couche d'abstraction, ainsi que le permet UML, et cette compréhension ne pourra que s'en trouver renforcée. Chacun de ces cinq langages offre des particularités amenant les praticiens de l'un ou l'autre à le prétendre supérieur aux autres : la puissance du C++, la compatibilité Windows et l'intégration XML de C#, l'anti-Microsoft et le leadership de Java en matière de serveurs d'entreprise, les vertus pédagogiques et l'aspect « scripting » de Python, le succès incontestable de PHP 5 pour la mise en place simplifiée d'une solution web dynamique et la capacité de s'interfacer aisément avec les bases de données. Nous nous désintéresserons ici complètement de ces querelles de clochers, a fortiori car notre projet pédagogique nous conduit bien davantage à nous pencher sur ce qui les réunit plutôt que ce qui les différencie. C'est leur multiplicité qui a présidé à cet ouvrage et qui en fait, nous l'espérons, son originalité. Nous n'allons pas nous en plaindre et défendons en revanche l'idée que le choix définitif de l'un ou l'autre de ces langages dépend davantage d'habitude, d'environnement professionnel ou d'enseignement, de questions sociales et économiques et surtout de la raison concrète de cette utilisation (pédagogie, performance machine, adéquation web ou base de données ...).

Quelques amabilités glanées dans *Masterminds of Programming*

Bjarne Stroustrup (créateur du C++) : « *J'avais prédit que s'il voulait percer, Java serait contraint de croître significativement en taille et en complexité. Il l'a fait.* »

Guido van Rossum (créateur de Python) : « *Je dis qu'une ligne de Python, de Ruby, de Perl ou de PHP équivaut à 10 lignes de Java.* »

Tom Love (co-créateur d'Objective-C, le langage OO de prédilection pour le développement des applications Apple et plus récemment iPhone, iPod et autres smartphones) : « *Tant Objective-C que C++ sont nés au départ du langage C. Dans le premier cas, ce fut un petit langage, simple, élégant, net et bien défini ; dans l'autre, ce fut une abomination hyper compliquée et présentant de véritables défauts de conceptions.* »

James Gosling (créateur de Java) : « *Les pointeurs en C++ sont un désastre, une véritable incitation à programmer de manière erronée* » et « *C# a tout pompé sur Java, à l'exception de la sûreté et de la fiabilité par la prise en charge de pointeurs dangereux qui m'apparaissent comme grotesquement stupides.* »

Anders Hejlsberg (créateur de C#) : « *Je ne comprends pas pourquoi Java a choisi de ne pas évoluer. Si vous regardez l'histoire de l'industrie, tout n'est qu'une question d'évolution. À la minute où vous arrêtez d'évoluer, vous signez votre arrêt de mort.* »

James Rumbaugh (un des trois concepteurs d'UML) : « *Je pense qu'utiliser UML comme générateur de code est une idée exécrationnelle. Il n'y a rien de magique au sujet d'UML. Si vous pouvez créer du code à partir des diagrammes, alors il s'agit d'un langage de programmation. Or UML n'est pas du tout conçu comme un langage de programmation.* »

Bertrand Meyer (créateur d'Eiffel et défenseur de la programmation OO dite par « contrats ») : « *Je ne comprends pas comment l'on peut programmer sans prendre le temps et la responsabilité de se demander ce que les éléments du programme ont la charge de faire. C'est une question à poser à Gosling, Stroustrup, Alan Kay ou Hejlsberg.* »

Récemment, un livre intitulé *Masterminds of Programming* (O'Reilly, 2009) et compilant un ensemble d'entretiens avec les créateurs des langages de programmation, nous a convaincu du bien-fondé de ce type de démarche comparative. Il apparaît en effet, au vu de la guerre des mots à laquelle se livrent ses créateurs, qu'aucun langage de programmation ne peut vraiment s'appréhender sans partiellement le comparer à d'autres. En fait, tous se positionnent dans une forme de rupture ou de remplacement par rapport à ses prédécesseurs.

Enfin, nous souhaitons que cet ouvrage, tout en restant suffisamment détaché de toute technologie, couvre l'essentiel des problèmes posés par la mise en œuvre des objets en informatique : leur stockage sur le disque dur, leur interfaçage avec les bases de données, leur fonctionnement en parallèle et leur communication à travers Internet. Ce faisant nous acceptons de perdre un peu en précision, et il nous apparaît nécessaire de mettre en garde le lecteur. Ce livre n'aborde aucune des technologies en profondeur, mais les approche toutes dans ce qu'elles ont de commun et qui se devrait de survivre pour des siècles et des siècles...

Plan de l'ouvrage

Les 23 chapitres de ce livre peuvent se répartir en cinq grandes parties.

Le **premier chapitre** constitue une partie en soi car il a pour importante mission d'initier aux briques de base de la programmation orientée objet, sans aucun développement technique : une première esquisse, teintée de sciences cognitives, et toute en intuition, des éléments essentiels de la pratique OO.

La **deuxième partie** intègre les quatorze chapitres suivants. Il s'agit pour chacun d'entre eux de décrire, plus techniquement cette fois, ces briques de base : objet, classe (**chapitres 2 et 3**), messages et communication entre objets (**chapitres 4, 5 et 6**), encapsulation (**chapitres 7 et 8**), gestion mémoire des objets (**chapitre 9**), modélisation objet (**chapitre 10**), héritage et polymorphisme (**chapitres 11 et 12**), classe abstraite (**chapitre 13**), clonage et comparaison d'objets (**chapitre 14**), interface (**chapitre 15**).

Chacune de ces briques est illustrée par des exemples en Java, C#, C++, Python, PHP 5 et UML. Nous y faisons le pari que cette mise en parallèle est la voie la plus naturelle pour la compréhension des mécanismes de base : extraction du concept par la multiplication des exemples.

La **troisième partie** reprend, dans le droit fil des ouvrages dédiés à l'un ou l'autre langage objet, des notions jugées plus avancées : les objets distribués, Corba, RMI, les services web (**chapitre 16**), le multithreading ou programmation parallèle (ou concurrentielle, **chapitre 17**), la programmation événementielle (**chapitre 18**) et enfin la sauvegarde des objets sur le disque dur, y compris l'interfaçage entre les objets et les bases de données relationnelles (**chapitre 19**). Là encore, le lecteur se trouvera le plus souvent en présence de plusieurs versions dans les cinq langages de ces mécanismes. La nouvelle plate-forme LINQ de Microsoft s'y trouve abordée, car elle propose une manière radicalement neuve de penser la mise en correspondance entre les objets et les informations stockées sous forme relationnelle ou XML.

La **quatrième partie** décrit plusieurs projets de programmation objet totalement aboutis, tant en UML qu'en Java. Elle inclut d'abord le **chapitre 20**, décrivant la modélisation objet d'un petit flipper et d'un

petit tennis, ainsi que les problèmes de conception orientée objet que cette modélisation pose. Le **chapitre 21**, lié au **chapitre 22**, décrit la manière dont les objets peuvent s'organiser en liste liée ou en graphe, mode de mise en relation et de regroupement des objets que l'on retrouve abondamment dans toute l'informatique. Le **chapitre 22** marie la chimie, la biologie et l'économie à la programmation OO. Il contient tout d'abord la programmation d'un réacteur chimique créant de nouvelles molécules à partir de molécules de base, tout en suivant à la trace l'évolution de la concentration des molécules dans le temps. La chimie – une chimie élémentaire acquise bien avant l'université – nous est apparue être une plate-forme pédagogique idéale pour l'assimilation des concepts objets. Nous proposons aussi des simulations élémentaires du système immunitaire et d'une économie de marché.

Enfin, la **dernière partie** se ramène au seul **chapitre 23**, dans lequel sont présentées plusieurs recettes de conception OO, résolvant de manière fort élégante un ensemble de problèmes récurrents dans la réalisation de programmes. Ces recettes de conception, dénommées *Design patterns*, sont devenues fort célèbres dans la communauté OO. Leur compréhension s'inscrit dans la suite logique de l'enseignement des briques et des mécanismes de base de l'OO. Elle fait souvent la différence entre l'apprenti et le compagnon parmi les programmeurs OO. Nous les illustrons en partie sur le flipper, la chimie, la biologie et l'économie des chapitres précédents.

À qui s'adresse ce livre ?

Cet ouvrage est sans nul doute destiné à un public assez large : industriels, enseignants et étudiants, mais sa vocation première n'en reste pas moins une initiation à la programmation orientée objet.

Ce livre sera un compagnon d'étude enrichissant pour les étudiants qui comptent la programmation objet dans leur cursus d'étude (et toutes technologies s'y rapportant : Java, C++, C#, Python, PHP, Corba, RMI, services web, UML, LINQ). Il devrait les aider, le cas échéant, à évoluer de la programmation procédurale à la programmation objet, pour aller ensuite vers toutes les technologies s'y rapportant.

Nous ne pensons pas, en revanche, que ce livre peut seul prétendre à constituer une porte d'entrée dans le monde de la programmation tout court. Comme dit précédemment, nous estimons qu'il est idéal d'aborder en même temps les mécanismes OO et procéduraux. Pour des raisons évidentes de place et car les librairies informatiques en regorgent déjà, nous avons fait l'impasse d'un enseignement de base des mécanismes procéduraux : variables, boucles, instructions conditionnelles, éléments fondamentaux et compagnons indispensables à l'assimilation de l'OO. Nous pensons, dès lors, que ce livre sera plus facile à aborder pour des lecteurs ayant déjà un peu de pratique de la programmation dite procédurale, et ce, dans un quelconque langage de programmation. Finalement, précisons que s'il ne prétend pas être exhaustif – et bien qu'à sa 6e édition – il résiste assez bien au temps. Ses pages ne jaunissent pas trop et, tout comme la plupart des technologies qu'il recense, il reste rétrocompatible avec ses versions précédentes. Il suit les évolutions de toutes ces technologies, même si celles-ci restent délibérément accrochées aux principes OO qui en font son sujet et, nous l'espérons, son attrait principal.

Table des matières

Avant-propos	V
L'orientation objet (OO) en quelques mots	VI
Les grands acteurs de l'orienté objet	... VIII
Objectifs de l'ouvrage	... IX
Plan de l'ouvrage	... XI
À qui s'adresse ce livre ?	... XII

CHAPITRE 1

Principes de base : quel objet pour l'informatique ?..... **1**

Le trio <entité, attribut, valeur>	... 2
Stockage des objets en mémoire	... 2
Types primitifs	... 4
Le référent d'un objet	... 5
Plusieurs référents pour un même objet	... 6
L'objet dans sa version passive	... 7
L'objet et ses constituants	... 7
Objet composite	... 8
Dépendance sans composition	... 9
L'objet dans sa version active	... 9
Activité des objets	... 9
Les différents états d'un objet	... 9
Les changements d'état : qui en est la cause ?	... 10
Comment relier les opérations et les attributs ?	... 10
Introduction à la notion de classe	... 11
Méthodes et classes	... 11
Sur quel objet précis s'exécute la méthode ?	... 13
Des objets en interaction	... 14
Comment les objets communiquent	... 14
Envoi de messages	... 15
Identification des destinataires de message	... 15

Des objets soumis à une hiérarchie	... 16
Du plus général au plus spécifique	... 16
Dépendance contextuelle du bon niveau taxonomique	... 17
Polymorphisme	... 18
Héritage bien reçu	... 19
Exercices	... 19

CHAPITRE 2

Un objet sans classe... n'a pas de classe..... **21**

Constitution d'une classe d'objets	... 22
Définition d'une méthode de la classe :	
avec ou sans retour	... 23
Identification et surcharge des méthodes par leur signature	... 24
La classe comme module fonctionnel	... 26
Différenciation des objets par la valeur des attributs	... 26
Le constructeur	... 26
Mémoire dynamique, mémoire statique	... 28
La classe comme garante de son bon usage	... 29
La classe comme module opérationnel	... 30
Mémoire de la classe et mémoire des objets	... 30
Méthodes de la classe et des instances	... 32
Un premier petit programme complet dans les cinq langages	... 32
En Java	... 33
En C#	... 35
En C++	... 37
En Python	... 40

En PHP 5	42	La compilation Java : effet domino	71
La classe et la logistique de développement	44	En C#, Python, PHP 5 ou C++	72
Classes et développement de sous-ensembles		De l'association unidirectionnelle	
logiciels	44	à l'association bidirectionnelle	74
Classes, fichiers et répertoires	45	Auto-association	77
Exercices	46	Paquets et espaces de noms	79
		Exercices	82
CHAPITRE 3		CHAPITRE 6	
Du faire savoir au savoir-faire... du procédural à l'OO	51	Méthodes ou messages ?.....	83
Objectif objet : les aventures de l'OO	52	Passage d'arguments prédéfinis	
Argumentation pour l'objet	52	dans les messages	84
Transition vers l'objet	53	En Java	84
Mise en pratique	53	<i>Résultat</i>	84
Simulation d'un écosystème	53	En C#	85
Analyse	54	<i>Résultat</i>	86
Analyse procédurale	54	En C++	86
Fonctions principales	54	<i>Résultat</i>	87
Conception	55	En Python	87
Conception procédurale	55	<i>Résultat</i>	88
Conception objet	56	En PHP 5	88
Conséquences de l'orientation objet	57	Passage d'argument objet dans les messages	91
Les acteurs du scénario	57	En Java	91
Indépendance de développement		<i>Résultat</i>	91
et dépendance fonctionnelle	57	En C#	92
Petite allusion (anticipée) à l'héritage	58	<i>Résultat</i>	94
La collaboration des classes deux à deux	58	En PHP 5	94
		En C++	96
		<i>Résultat passage par valeur</i>	97
CHAPITRE 4		<i>Résultat passage par référent</i>	97
Ici Londres : les objets parlent aux objets.....	59	En Python	97
Envois de messages	60	<i>Résultat</i>	98
Association de classes	61	Une méthode est-elle d'office un message ?	98
Dépendance de classes	64	Même message, plusieurs méthodes	98
Réaction en chaîne de messages	66	Nature et rôle, type, classe et interface :	
Exercices	67	quelques préalables	99
		Interface : liste de signatures de méthodes	
CHAPITRE 5		disponibles	100
Collaboration entre classes ..	69	Des méthodes strictement internes	101
Pour en finir avec la lutte des classes	70	La mondialisation des messages	101

Message sur Internet	101		
L'informatique distribuée	102		
Exercices	103		
CHAPITRE 7			
L'encapsulation			
des attributs	107		
Accès aux attributs d'un objet	108		
Accès externe aux attributs	108		
Cachez ces attributs que je ne saurais voir	109		
Encapsulation des attributs	110		
<i>En Java</i>	110		
<i>En C++</i>	111		
<i>En C#</i>	111		
<i>En PHP 5</i>	112		
<i>En Python</i>	113		
Encapsulation : pour quoi faire ?	114		
Pour préserver l'intégrité des objets	114		
La gestion d'exception	115		
Pour cloisonner leur traitement	117		
Pour pouvoir faire évoluer leur traitement en douceur	117		
La classe : enceinte de confinement	119		
Exercices	119		
CHAPITRE 8			
Les classes et leur jardin			
secret	121		
Encapsulation des méthodes	122		
Interface et implémentation	122		
Toujours un souci de stabilité	123		
Signature d'une classe : son interface	125		
Les niveaux intermédiaires d'encapsulation	126		
<i>Classes amies</i>	126		
<i>Une classe dans une autre</i>	127		
<i>Utilisation des paquets</i>	128		
Afin d'éviter l'effet papillon	130		
Exercices	132		
CHAPITRE 9			
Vie et mort des objets 135			
Question de mémoire	136		
Un rappel sur la mémoire RAM	136		
L'OO coûte cher en mémoire	137		
Qui se ressemble s'assemble : le principe de localité	137		
Les objets intermédiaires	138		
Mémoire pile	138		
<i>En C++</i>	140		
<i>En C#</i>	141		
Disparaître de la mémoire comme de la vie réelle	144		
Mémoire tas	145		
C++ : le programmeur est le seul maître à bord	145		
La mémoire a des fuites	146		
En Java, C#, Python et PHP 5 : la chasse au gaspi	148		
En Java	149		
En C#	150		
En PHP 5	150		
Le ramasse-miettes (ou garbage collector)	151		
Des objets qui se mordent la queue	152		
En Python	153		
Exercices	155		
CHAPITRE 10			
UML 2 159			
Diagrammes UML 2	161		
Représentation graphique standardisée	162		
Du tableau noir à l'ordinateur	163		
Programmer par cycles courts en superposant les diagrammes	164		
Diagramme de classe et diagramme de séquence	165		
Diagramme de classe	166		
Une classe	166		
<i>En Java : UML1.java</i>	166		
<i>En C# : UML1.cs</i>	167		

<i>En C++ : UML1.cpp</i>	168	Diagramme d'états-transitions	211
<i>En Python : UML1.py</i>	169	Exercices	214
<i>En PHP 5 : UML1.php</i>	169		
Similitudes et différences entre les langages .	170	CHAPITRE 11	
Association entre classes	170	Héritage..... 221	
<i>En Java : UML 2.java</i>	171	Comment regrouper les classes	
<i>En C# : UML 2.cs</i>	171	dans des superclasses ?	222
<i>En C++ : UML 2.cpp</i>	172	Héritage des attributs	223
<i>En Python : UML 2.py</i>	173	Pourquoi l'addition de propriétés ?	226
<i>En PHP 5 : UML2.php</i>	174	L'héritage : du cognitif aux taxonomies	226
Similitudes et différences entre les langages .	174	Interprétation ensembliste de l'héritage	227
Pas d'association sans message	175	Qui peut le plus peut le moins	228
Rôles et cardinalité	176	Héritage ou composition ?	229
Dépendance entre classes	184	Économiser en ajoutant des classes ?	230
Composition	185	Héritage des méthodes	230
En Java	186	Code Java	233
<i>UML3.java</i>	187	Code C#	234
<i>UML3bis.java</i>	189	Code C++	235
En C#	190	Code Python	237
<i>UML3.cs</i>	191	Code PHP 5	238
<i>UML3bis.cs</i>	192	La recherche des méthodes	
En C++	193	dans la hiérarchie	239
<i>UML3.cpp</i>	194	Encapsulation protected	240
<i>UML3bis.cpp</i>	196	Héritage et constructeurs	241
En Python	197	Premier code Java	241
<i>UML3.py</i>	197	Deuxième code Java	242
<i>UML3bis.py</i>	198	Troisième code Java : le plus logique	
En PHP 5	199	et le bon	243
Classe d'association	200	En C#	244
Les paquets	201	En C++	245
Les bienfaits d'UML	202	En Python	246
Un premier diagramme de classe		En PHP 5	246
de l'écosystème	202	Héritage public en C++	247
Des joueurs de football qui font		Le multihéritage	248
leurs classes	202	Ramifications descendantes et ascendantes	248
Les avantages des diagrammes de classes .	202	Multihéritage en C++ et Python	249
Un diagramme de classe simple à faire, mais		<i>Code C++ illustrant le multihéritage</i>	250
qui décrit une réalité complexe à exécuter .	204	<i>Code Python illustrant le multihéritage</i>	251
Procéder de manière modulaire		Des méthodes et attributs portant un même	
et incrémentale	205	nom dans des superclasses distinctes	252
Diagramme de séquence	205		

Code C++ illustrant un premier problème lié au multihéritage	252
En Python	254
Plusieurs chemins vers une même superclasse	255
Code C++ : illustrant un deuxième problème lié au multihéritage	256
L'héritage virtuel	258
Exercices	259

CHAPITRE 12

Redéfinition des méthodes 265

La redéfinition des méthodes	266
Beaucoup de verbiage mais peu d'actes véritables	267
Un match de football polymorphique ...	268
La classe Balle	269
En Java	269
En C++	270
En C#	270
En Python	270
En PHP 5	270
La classe Joueur	270
En Java	270
En C++	271
En C#	272
En Python	273
En PHP 5	274
Précisons la nature des joueurs	274
En Java	275
En C++	276
En C#	277
En Python	278
En PHP 5	279
Passons à l'entraîneur	280
En Java	280
En C++	281
En C#	281
En Python	282
En PHP 5	282
Passons maintenant au bouquet final ...	282

En Java	282
Un même ordre mais une exécution différente	284
C++ : un comportement surprenant	285
Polymorphisme : uniquement possible dans la mémoire tas	289
En C#	289
En Python	291
En PHP 5	292
Quand la sous-classe doit se démarquer pour marquer	293
Les attaquants participent à un casting ...	294
Éviter les « mauvais castings »	295
En C++	296
En C#	296
Le casting a mauvaise presse	297
Redéfinition et encapsulation	298
Exercices	299

CHAPITRE 13

Abstraite, cette classe est sans objet..... 311

De Canaletto à Turner	312
Des classes sans objet	312
Du principe de l'abstraction à l'abstraction syntaxique	313
Classe abstraite	315
new et abstract incompatibles	316
Abstraite de père en fils	316
Un petit exemple dans quatre langages de programmation	317
En Java	317
En C#	318
En PHP 5	319
En C++	320
L'abstraction en Python	321
Un petit supplément de polymorphisme ..	322
Les enfants de la balle	322
Cliquez frénétiquement	322
Le Paris-Dakar	324
Le polymorphisme en UML	325

Exercices	326
<i>Exercice 13.11</i>	335
<i>Exercice 13.12</i>	335

CHAPITRE 14

Clonage, comparaison et affectation d'objets 339

Introduction à la classe Object	340
Une classe à compétence universelle	341
Code Java illustrant l'utilisation de la classe Vector et innovation de Java 5	341
<i>Nouvelle version du code</i>	342
Décortiquons la classe Object	343
Test d'égalité de deux objets	345
Code Java pour expérimenter la méthode equals(Object o)	345
Égalité en profondeur	348
Le clonage d'objets	350
Code Java pour expérimenter la méthode clone()	350
Égalité et clonage d'objets en Python	354
Code Python pour expérimenter l'égalité et le clonage	354
Égalité et clonage d'objets en PHP 5	355
Code PHP 5 pour expérimenter l'égalité et le clonage	355
Égalité, clonage et affectation d'objets en C++	357
Code C++ illustrant la duplication, la comparaison et l'affectation d'objets	357
Traisons d'abord la mémoire tas	361
Surcharge de l'opérateur d'affectation	363
Comparaisons d'objets	363
La mémoire pile	364
Surcharge de l'opérateur de comparaison	364
Dernière étape	365
Code C++ de la classe O1 créé automatiquement par Rational Rose	366
En C#, un cocktail de Java et de C++	368
Pour les structures	368

Pour les classes	368
Code C#	368
Exercices	374

CHAPITRE 15

Interfaces 375

Interfaces : favoriser la décomposition et la stabilité	376
Java, C# et PHP 5 : interface et héritage	377
Les trois raisons d'être des interfaces	378
Forcer la redéfinition	378
Code Java illustrant l'interface Comparable	379
Code Java illustrant l'interface ActionListener	381
Code Java illustrant l'interface KeyListener	382
Permettre le multihéritage	384
La carte de visite de l'objet	385
Code Java	386
Code C#	387
Code PHP 5	391
Les interfaces dans UML 2	392
En C++ : fichiers .h et fichiers .cpp	393
Interfaces : du local à Internet	397
Exercices	397

CHAPITRE 16

Distribution gratuite d'objets : pour services rendus sur le réseau..... 401

Objets distribués sur le réseau : pourquoi ?	402
Faire d'Internet un ordinateur géant	402
Répartition des données	403
Répartition des utilisateurs et des responsables	403
Peer-to-peer	404
L'informatique ubiquitaire	405
Robustesse	406
RMI (Remote Method Invocation)	406
Côté serveur	407
Côté client	408

RMIC : stub et skeleton	410	L'effet du multithreading sur les diagrammes de séquence UML	450
Lancement du registre	411	Du multithreading aux applications distribuées	451
Corba (Common Object Request Broker Architecture)	412	Des threads équirépartis	452
Un standard : ça compte	413	En Java	452
IDL	413	En C#	452
Compilateur IDL vers Java	414	En Python	453
Côté client	415	Synchroniser les threads	454
Côté serveur	417	En Java	454
Exécutons l'application Corba	418	En C#	456
Corba n'est pas polymorphique	419	En Python	459
Ajoutons un peu de flexibilité à tout cela	420	Exercices	462
Corba : invocation dynamique versus invocation statique	421		
Jini	421		
XML : pour une dénomination universelle des services	422		
Les services web sur .Net	425	CHAPITRE 18	
Code C# du service	425	Programmation	
WDSL	426	événementielle	465
Création du proxy	427	Des objets qui s'observent	466
Code C# du client	428	En Java	467
Soap (Simple Object Access Protocol)	428	La plante	467
Invocation dynamique sous .Net	429	Du côté du prédateur	468
Invocation asynchrone en .Net	430	Du côté de la proie	469
Mais où sont passés les objets ?	432	Finalement, du côté de la Jungle	469
Un annuaire des services XML universel : UDDI	435	Résultat	470
Services web versus RMI et Corba	435	En C# : les délégués	470
Services web versus Windows Communication Foundation (WCF)	436	Généralités sur les délégués dans .Net	470
Exercices	436	Retour aux observateurs et observables	474
		<i>Tout d'abord, la plante</i>	474
		<i>Du côté du prédateur</i>	475
		<i>Du côté de la proie</i>	476
		<i>Finalement, du côté de la Jungle</i>	476
		En Python : tout reste à faire	479
		Un feu de signalisation plus réaliste	481
		En Java	482
		Exercices	484
CHAPITRE 17		CHAPITRE 19	
Multithreading	439	Persistance d'objets	485
Informatique séquentielle	441	Sauvegarder l'état entre deux exécutions	486
Multithreading	443	Et que dure le disque dur	486
Implémentation en Java	444		
Implémentation en C#	446		
Implémentation en Python	449		

Quatre manières d'assurer la persistance des objets	486
Simple sauvegarde sur fichier	487
Utilisation des streams ou flux	487
Qui sauve quoi ?	488
En Java	488
En C#	490
En C++	491
En Python	492
En PHP 5	493
Sauvegarder les objets sans les dénaturer :	
la sérialisation	494
En Java	495
En C#	497
En Python	498
Contenu des fichiers de sérialisation :	
illisible	499
Les bases de données relationnelles	499
SQL	500
Une table, une classe	500
Comment interfacier Java et C#	
aux bases de données	502
<i>En Java</i>	503
<i>En C#</i>	505
Relations entre tables et associations	
entre classes	507
<i>Relation 1-n</i>	507
<i>Relation n-n</i>	510
<i>Dernier problème : l'héritage</i>	511
Réservation de places de spectacles	512
Les bases de données relationnelles-objet	517
SQL3	519
Les bases de données orientées objet	521
OQL	522
Django et Python	523
LINQ	524
Premier exemple de LINQ agissant sur une collection d'objets	525
Second exemple de LINQ agissant sur une base de données relationnelle	527
Exercices	531

CHAPITRE 20

Et si on faisait un petit flipper ? 533

Généralités sur le flipper et les GUI	534
Une petite animation en C#	540
Retour au Flipper	545
Code Java du Flipper	548
Un petit tennis	558

CHAPITRE 21

Les graphes 565

Le monde regorge de réseaux	566
Tout d'abord : juste un ensemble d'objets	568
Liste liée	570
En Java	572
En C++	574
La généricité en C++	576
La généricité en Java et C#	580
Passons aux graphes	586
Exercices	590

CHAPITRE 22

Petites chimie, biologie et économie OO amusantes 595

Pourquoi de la chimie OO ?	596
Chimie computationnelle	596
Chimie comme plate-forme didactique	596
Une aide à la modélisation chimique	596
Les diagrammes de classes du réacteur chimique	597
La classe Composant_Chimique	598
<i>Les classes Composant_Neutre et Composant_Charge</i>	599
<i>Les trois sous-classes de composants neutres</i>	600
<i>Les trois sous-classes de composants chargés</i>	602
La classe NœudAtomique	603
La classe NœudMoléculaire	603
La classe Liaison	603
Le graphe moléculaire	604
<i>Les règles de canonisation</i>	606

Les réactions chimiques	607	Dernière simulation : économie de marché	617
<i>Une première réaction de croisement</i>	607		
<i>Une autre réaction de croisement</i>			
<i>un peu plus sophistiquée</i>	608		
<i>Une réaction d'ouverture de liaison</i>	608		
<i>Réaction de transfert de charge</i>	609		
<i>Réaction de type Ion-Molécule</i>	609		
<i>Comment est calculée la cinétique</i>			
<i>réactionnelle</i>	609		
<i>La classe Reaction</i>	610		
<i>Les sous-classes de Reaction</i>	610		
Quelques résultats du simulateur	610		
<i>Première simulation : une seule molécule</i>			
<i>produite</i>	611		
<i>Deuxième simulation : plusieurs molécules</i>			
<i>produites</i>	612		
<i>Troisième simulation : De nombreuses</i>			
<i>molécules complexes produites</i>	612		
<i>Pourquoi un tel simulateur ?</i>	613		
La simulation immunologique en OO ? .	613		
Petite introduction au fonctionnement			
du système immunitaire	613		
Le diagramme UML d'états-transitions .	615		
		CHAPITRE 23	
		Design patterns	621
		Introduction aux design patterns	622
		De l'architecte à l'archiprogrammeur	622
		Les patterns « trucs et ficelles »	624
		Le pattern Singleton	625
		Le pattern Adaptateur	626
		Les patterns Patron de méthode,	
		Proxy, Observer et Memento	627
		Le pattern Flyweight	628
		Les patterns Builder et Prototype	629
		Le pattern Façade	631
		Les patterns qui se jettent à l'OO	632
		Le pattern Command	633
		Le pattern Décorateur	636
		Le pattern Composite	639
		Le pattern Chain of responsibility	639
		Les patterns Strategy, State et Bridge	640
		Index	645

Index

Symboles

.Net 142, 390, 423, 506

Numeriques

2-tiers 403

3-tiers 403

A

abstract 316

Abstract Factory 623

abstraction 312

Access 500

acteur 57

ActionListener en Java 379

adressage indirect 6, 92

affectation d'objet 362, 363

agrégation 185

analyse

objet 56

procédurale 54

annuaire 421, 435

argument 64, 184

objet 91, 140

passage par référent 86, 89

assemblage 79, 128

association de classes 61, 176

atome 600

attribut 2, 61, 62

private 109, 110

automate cellulaire 442

B

base 278

base de données 403, 454, 499

objet 60

OO 521, 523, 524

relationnelle 5, 499

relationnelle-objet 60, 512,
517

Berners-Lee, Tim 423

biologie 60, 442

Booch, Gary 160

Bridge 623

C

cardinalité 176

cas d'utilisation 160

casting 90, 294, 341, 496

critique 297

explicite 228

implicite 228, 294

cerveau 441

charte du bon programmeur

OO 108, 130

chimie

computationnelle 596

cinétique réactionnelle 609

classe

abstraite 268, 313, 315, 321,
377

association 74, 176

dépendance 91, 184

enceinte de confinement 119

fichier 65, 72, 184

généralité 12

interaction 70

Object 340, 343, 496, 580

observable 466, 467

sans objet 312

Thread 445

Vector 341, 570

clé

étrangère 508

primaire 501, 502

clonage d'objet 98, 350

CLR 143

CLS 143

collection 230

comparaison d'objets 363

compilation 61, 70, 71, 109,

228, 239, 254, 293, 294,

312, 379, 387, 579

Composite 623

composition 200

d'objets 9, 185, 229

compteur de référents 151

constructeur

héritage 244

par copie 98, 141, 362, 367

cookie 434

Corba 160, 406, 412, 422

invocation

dynamique 421

statique 421

services 416

création automatique de

code 203

D

Dahl, Ole-Johan 52

Deittel et Deittel 71

délégué 470

en C# 448

delete 145, 148, 193, 210

dépendance

de classe 64, 184

design pattern 411, 466

destructeur 141, 190, 193, 195

diagramme

- d'objet 183
- de classe 161, 165, 202, 222, 267, 325, 385, 571
- de classes 597
- de composant 184, 393
- de séquence 161, 165, 205, 211, 450
- UML 161
- disque dur 65, 136, 486
- dll 72
- DTD (Document Type Definition) 424
- E**
- écosystème 53, 202, 222, 313, 439
- effet papillon 132
- égalité de deux objets 345
- encapsulation 100, 110, 114, 130, 240, 241, 377
- des méthodes 122
- espace de nommage 129, 414
- F**
- Façade 623
- Factory Method 623
- fichiers
 - .cpp en C++ 393
 - .h en C++ 393
- finalize 149, 186
- flux 487
- flux filtré 487
- fonctionnement de type
 - yoyo 240
- friend 126, 364
- fuite de mémoire 146
- G**
- Gamma, Helm, Johnson et Vlissides 623
- Gang des quatre 411, 623
- garbage collector 151, 155
- Gates, Bill 142, 324
- Gemstone 521
- généricité 90, 576
- gestion
 - d'exception 117, 295, 352, 406, 490
 - mémoire 139, 144, 174
- Gnutella 404
- Gosling, James 70
- graphe 566, 586
 - moléculaire 604
- H**
- Hejlsberg, Anders 142
- héritage
 - addition de propriétés 226
 - arbre versus graphe 248
 - d'interfaces 377
 - des attributs 223
 - des flux 487
 - des méthodes 230
 - économie 230
 - généralités 17, 58, 202, 511
 - interprétation
 - ensembliste 227, 276
 - justification 230, 267
 - mécanisme 224
 - public en C++ 247
 - versus composition 229, 538
 - virtuel en C++ 258
- Hopfield, John 567
- HP 425
- HTML 424
- HTTP 423
- I**
- IBM 414, 425
- IDL (Interface Definition Language) 413
- IDLJ (Interface Definition Language Java) 414
- imbrication de classe 127, 190
- include 395
- indépendance logicielle 58, 117
- informatique
 - séquentielle 441
 - ubiquitaire 405
- Informix 500
- instance 12
- intégrité référentielle 508
- intelligence artificielle 226, 227
- interface 100, 122, 125, 352, 376, 407, 414, 496
 - fichier 385
 - graphique 322
 - Observer 468
 - versus implémentation 122, 376
 - versus implémentation en C++ 394
- Internet 101, 397, 402, 451
- Iona 414
- J**
- Jacobson, Ivar 160
- Java 70
- JAX-RPC 423
- JDBC 504
- Jini 71, 421, 435
- Jobs, Steve 323
- JXTA 405
- K**
- Kant 227
- Kauffman, Stuart A. 130, 597
- Kay, Alan 323, 324
- Kazaa 404
- KeyListener en Java 379, 382
- Khün, Thomas 227
- L**
- leasing 422
- liaison 603
- LIFO (Last-In First-Out) 139
- LISP 153
- liste liée 570
- lookup 421
- M**
- machine de von Neumann 442
- match de football 202, 268
- Memento 623

- mémoire
 - associative 567
 - cache 137
 - compactage 153
 - fuite 146
 - pile 85, 138, 170, 174, 191, 193, 236, 364, 569
 - RAM 10, 136, 137, 240, 357, 486
 - tas 145, 170, 174, 193, 236, 289, 321, 361, 569
- message 98, 175
 - asynchrone 451
 - envoi 66, 170, 405, 419
 - généralités 15, 60
 - synchrone 451
- méthode
 - abstraite 315, 377
 - appel 13, 77
 - argument 64
 - d'accès 112
 - définition 11, 14
 - main 60
 - message 98
 - private 122
 - public 122
 - redéfinition 266
 - run 446
 - signature 98
 - virtuelle 288
 - pure 315
 - Meyer, Bertrand 66, 143
 - Microsoft 160, 423
 - Minsky, Marvin 226, 227, 567
 - molécule 600
 - MouseListener en Java 378
 - multihéritage 248, 384, 546
 - d'interfaces 255
 - en C++ 249
 - multitâche 443
 - multithreading 206, 343, 379
 - définition 443
 - généralités 139
 - join 461
 - par composition 448
 - par héritage 448
 - priorité 452
 - sleep 452
 - start 448
 - suspension 452
 - synchronisation 451, 454
 - multi-tiers 403
- N**
- Napster 404
- new 137, 145, 312, 316
- nœud 567
- Nygaard, Kristen 52
- O**
- ObjectStore 521
- objet
 - affectation 362
 - clonage 350, 354, 355
 - composite 8
 - composition 62
 - cycle de vie 10, 144
 - dépendance 466
 - distribué 102, 402, 406, 451
 - état 9
 - intégrité 114
 - interaction 14, 58, 60, 125
 - persistance 486
 - réfèrent 5
 - sauvegarde 367
 - versus procédural 78, 117, 144
- Observer 623
- ODBC (Open DataBase Connectivity) 504
- ODMG (Object Data Management Group) 522
- OleDb 506
- OMG (Objet Management Group) 160, 413
- OMT (Object Modelling Technique) 160
- OOD (Object Oriented Design) 160
- OOSE (Object Oriented Software Engineering) 160
- OQL (Objet Query Language) 518, 522
- Oracle 500, 519
- OrbixWeb 414
- override 275, 278, 319, 389
- P**
- package 128, 414
- parallélisme 442
- passage
 - par réfèrent 96
 - par valeur 85, 96
- pattern
 - Abstract Factory 623
 - Bridge 623
 - Façade 623
 - Factory Method 623
 - Observer 623
 - Proxy 623
 - Singleton 623
- peer-to-peer 404
- Piaget, Jean 227
- Poet 521
- pointeur 86, 140, 170
- polymorphisme 255, 268, 389, 545
 - Corba 419
 - et casting 298
 - généralités 18, 98, 322
 - mécanisme 267, 293
 - par défaut 285
 - RMI 420
- principe
 - de localité 137
 - de substitution 226, 228, 284, 294
- private 109, 117, 126
- programmation
 - événementielle 466

protected 128
 Proxy 623
 proxy 421, 427
 public 109, 117, 126

R

ramasse-miettes 92, 151, 155, 186, 195, 365
 Rational 160
 Rose 160, 175, 249, 365
 réacteur chimique 597
 réactions chimiques 607
 redéfinition des méthodes 275, 378
 référent 5, 91, 145, 149, 170, 174, 289, 347, 387, 448
 en C++ 86
 fou 146, 148
 registre 408, 411
 relation
 1-n 507
 n-n 510
 répertoire 81
 réseau 566
 de Hopfield 567
 de neurones 227, 442, 566
 génétique 566
 immunitaire 566
 réutilisation 230
 Riel, Arthur J. 108
 RMI (Remote Method Invocation) 70, 403, 406, 422
 robustesse 406
 rôles 176
 RTTI (Run-Time Type Information) 272
 Rumbaugh, James 160
 Runnable en Java 379
 RUP (Rational Unified Process) 165

S

sauvegarde d'objet 367

schéma XML 424
 sciences cognitives 19, 567
 sémaphore 458
 sérialisation 494
 services web 143, 423, 425
 seti@home 402
 signature 255, 376
 de méthode 122
 Simula 52, 323
 Singleton 623
 skeleton 410, 411
 Smalltalk
 voir aussi Squeak 323
 Soap 427, 428
 sous-classe 224
 SQL 500
 SQL3 518, 519
 Squeak
 voir aussi Smalltalk 324
 squelette de code 203
 stabilité du logiciel 117, 119, 123, 377, 397
 Stroustrup, Bjarne 89, 297, 576
 structure en C# 142, 170, 190, 291, 368, 390
 stub 410, 411, 415
 Sun 402, 423, 425
 super 244, 276
 superclasse 222
 surcharge d'opérateur 361, 364, 492
 Sybase 500
 synchronisation des threads 454
 système
 complexe 60, 130, 324, 567
 d'exploitation 322, 443

T

table 500
 virtuelle en C++ 288
 tableau 568
 d'objets 314, 341
 en C++ 281
 taxonomie 225, 267

Taylor, David A. 60
 TCP/IP 397
 template 577, 579
 temporisation 487
 thread 443
 TogetherJ 165, 203, 207, 249
 traitement en surface et en profondeur 357
 trio 2
 trois amigos 160
 typage
 dynamique 281, 287, 293, 312
 statique 281, 286, 293
 statique versus typage dynamique 284, 318
 type
 entier 12
 primitif 3, 62

U

UDDI (Universal Description Discovery and Integration) 435
 UML
 avantages 202
 UML (Unified Modelling Language) 160, 162, 164, 230, 325, 365, 450, 579, 596
 diagramme d'états-transitions 615
 UML 2
 interface 392

V

valeur 89
 Van Rossum, Guido 340
 Versant 521
 vie artificielle 442
 virtual 272, 273, 286, 389
 virtual/override 291
 Visibroker 414
 Visitor 623

von Neumann, John 442

W

W3 423

Walter Fontana 596, 597

WDSL 425, 435

Web sémantique 423

WebSphere 414

Windows 324

Wittgenstein 18

X

Xerox PARC 324

XML 422, 424